A Method for Automated Detection of Usability Problems from Client User Interface Events

Gilan M. Saadawi MD, PhD[1], Elizabeth Legowski[2], Olga Medvedeva, MS[2], Girish Chavan, MS[2]
and Rebecca S. Crowley MD, MS[1,2,3]

[1] Center for Biomedical Informatics, University of Pittsburgh School of Medicine, Pittsburgh PA
[2]Centers for Oncology and Pathology Informatics, University of Pittsburgh School of Medicine, Pittsburgh PA
[3] Intelligent Systems Program, University of Pittsburgh, Pittsburgh PA

ABSTRACT

Think-aloud usability analysis provides extremely useful data but is very time-consuming and expensive to perform because of the extensive manual video analysis that is required. We describe a simple method for automated detection of usability problems from client user interface events for a developing medical intelligent tutoring system. The method incorporates (1) an agent-based method for communication that funnels all interface events and system responses to a centralized database, (2) a simple schema for representing interface events and higher order subgoals, and (3) an algorithm that reproduces the criteria used for manual coding of usability problems. A correction factor was empirically determining to account for the slower task performance of users when thinking aloud. We tested the validity of the method by simultaneously identifying usability problems using TAU and manually computing them from stored interface event data using the proposed algorithm. All usability problems that did not rely on verbal utterances were detectable with the proposed method.

INTRODUCTION

Usability is the extent to which a system enables users, in a given context of use, to achieve specified goals effectively and efficiently. [1] Usability evaluation (UE) as defined by Nielsen [2], consists of methodologies for measuring usability attributes of a system user interface (UI). Different approaches to usability evaluation have been described, including empirical methods (such as think-aloud usability testing) and analytic methods (such as GOMS, Heuristic Evaluation, and Cognitive Walkthrough) [3].

User-based evaluations using think-aloud protocol (TAU) is generally considered to be the most valuable usability method since it yields the highest number of relevant usability problems [3]. In TAU, real participants are studied under laboratory or field conditions [2, 5]. Typically these studies involve (1) creation of a set of tasks, (2) video capture of the user attempting to complete the tasks using the system while they think aloud, and then (3) coding and analysis of the resulting data – both video and think-aloud using a metric for determining the existence of a usability problem. Analysis of the collected empirical data is considered the most resource demanding activity in a usability study. Not only is it time consuming, but evaluators may find themselves influencing the findings through different interpretations [6].

In contrast, automated usability methods target analysis of user interface events (UI events) that are generated as natural products of the normal operation of user interface systems. Because such events can be captured and because they indicate user behavior with respect to an application's user interface, they have long been regarded as a potentially fruitful source of information regarding application usage and usability. Capture and analysis of task times, percentage of task completion, error rates, duration and frequency of help usage provide excellent data for quantitatively characterizing on-line behavior and performance attributes. They are most useful in understanding user behavior and performance, comparing design alternatives and computing usability metrics [4].

However, user interface events are typically extremely voluminous and rich in detail. Consequently, one of the most challenging problems is to capture, store and retrieve data at a level of abstraction that is useful to investigators interested in analyzing application usage or evaluating usability [4].

Automation has been used predominantly in two ways within usability testing: (1) automated capture of use data and (2) automated analysis of these data according to metrics [6]. Automatic capture of UIE has several potential advantages over traditional TAU. First, it avoids the cost and time required to perform TAU sessions and coding of the data. Second, it can provide real time data to developers as new features are added. Third, it allows incorporation of multiple evaluations within the UI development phase [7].

In this manuscript we describe our existing system for automated capture of interface events, and test a method for automated usability analysis (AUA) based on classical usability evaluation. The test-bed for this work is SlideTutor – an Intelligent Tutoring System for visual diagnosis.

TEST-BED SYSTEM DESCRIPTION

SlideTutor [8] is a developing, web-deployed, Intelligent Tutoring System (ITS) designed for use by medical students and residents in Pathology and Dermatology. SlideTutor uses a set of novel interface elements, including a virtual microscope system and a reasoning interface, in which each student builds an argument for a particular microscopic diagnosis. The system is based on the Cognitive Tutor paradigm, in which each student action is evaluated by the tutoring system. Incorrect actions match against production rules that detect different types of errors-and provide context specific remediation. Correct actions are allowed, and move the students forward in the problem-solving environment. Requests for help generate responses from the tutor based on the next best step in the expert model.

In many respects, SlideTutor is an ideal system in which to deploy an automated module for detecting usability problems: (1) like other ITS, the highly interactive nature of the system requires constant attention to the usability of the UI, (2) the system changes frequently as new interfaces are developed for particular experiments, (3) TAU studies have become commonplace in our laboratory but are time-consuming to analyze, and (4) the client-server nature could be exploited to capture all events from any client to a single database.

COMMUNICATIONS AND LOGGING DATABASE

Communication among all modules of the system is agent-based, using the Agent Communication Language - a message exchange interaction protocol standard number developed by the Foundation for Intelligent Physical Agents (FIPA)[9]. The message format can fit an unlimited number of possible parameters. The corresponding database representation is generic and could be applied to virtually any interactive system. We use one database system (implemented with Oracle 9i) for managing all data generated during project experiments. The relational design allows us to store information regarding relationships among experiments, users, tutor cases, goals to achieve in each case, student action events and tutor response events. All events are time-stamped.

The events relevant to automated usability analysis include three categories. *Interface Events* record low-level human-computer interaction such as pressing a button or selecting a menu item. *Client Event*s capture combinations of *Interface Events* that represent the most atomic discrete subgoal, such as creating a hypothesis, identifying a feature, or asking for a hint. *Client Event*s are answered by *Tutor Response*s. *TutorResponse (TR)* indicate the response of the system to the last student

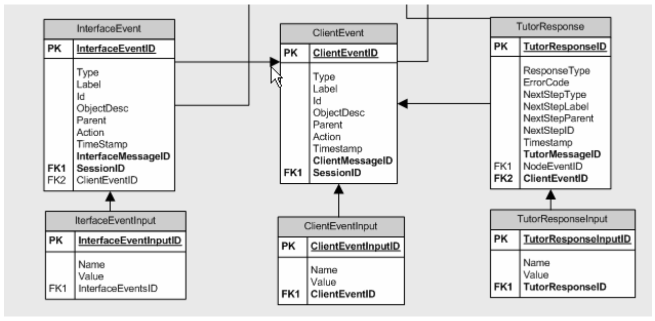

Figure 1. Interface Events, Client Events, and Tutor Responses

CETYPE	CELABEL	CEACTION	CETIMESTAMP	IETYPE	IELABEL	IEACTION	IETIMESTAMP
Finding	blister	Evidence	2005-04-13 16:25:37.024	Button	finding	Pressed	2005-04-13 16:25:22.44
Finding	blister	Evidence	2005-04-13 16:25:37.024	FINDINGTree	FINDING	Open	2005-04-13 16:25:23.912
Finding	blister	Evidence	2005-04-13 16:25:37.024	FINDINGTree	[FEATURE, DERMAL CHANGES]	TreeExpanded	2005-04-13 16:25:30.281
Finding	blister	Evidence	2005-04-13 16:25:37.024	FINDINGTree	[FEATURE, DERMAL CHANGES]	TreeSelected	2005-04-13 16:25:30.381
Finding	blister	Evidence	2005-04-13 16:25:37.024	FINDINGTree	[FEATURE, INTRAEPIDERMAL	TreeExpanded	2005-04-13 16:25:34.177
Finding	blister	Evidence	2005-04-13 16:25:37.024	FINDINGTree	[FEATURE, INTRAEPIDERMAL	TreeSelected	2005-04-13 16:25:34.237
Finding	blister	Evidence	2005-04-13 16:25:37.024	FINDINGTree	[FEATURE, INTRAEPIDERMAL	TreeSelected	2005-04-13 16:25:36.38
CETYPE	CELABEL	CEACTION	CETIMESTAMP	IETYPE	IELABEL	IEACTION	IETIMESTAMP

Table 1: One *Client Event* composed of multiple *Interface Events* (extract from protocol database).

action including the type of error for incorrect actions and the best next action at this point in the problem space. The part of the schema related to these three events is shown in Figure 1.

All *Interface Events (IE)* and *Client Events (CE)* can be represented as a simple aggregate of type, label and action. *Interface Event* types (IE_TYPE) indicates interface element class (e.g. button, menu). *Interface Event* Label (IE_LABEL) indicates the instance of the element (e.g. finding button, file menu). *Interface Event* Action (IE_ACTION) indicates the user action on the element (e.g. press, release, select). A single IE_TYPE can have multiple IE_LABEL and multiple IE_ACTION. A Button could be a Feature-Button or a Hypothesis-Button or a Hint-Button. And, the Feature-Button could have IE_ACTION of Button-Pressed, Button-Next or Button-Released.

Client Event represent tutor-respondable user actions, reflecting a single small subgoal, such as asking for a hint, creating a finding, or deleting a hypothesis. *Client Event* types (CE_TYPE) indicate sub-goal class (e.g. finding, hypothesis). Client Event Label (CE_LABEL) represents the instance of the subgoal (e.g. finding of blister, hypothesis of IgA Dermatosis), and Client Event Action indicates the action aspect of the subgoal (e.g. assert evidence, delete).

As an example, take the action of asserting a feature of blister in SlideTutor. To correctly assert this feature the student must: press the finding button, click on a region of the virtual slide, expand and select through a hierarchical tree of findings which opens in a pop-up window, and select the appropriate finding describing the region which was indicated in the image. Table 1 shows the single Client Event (see uniform timestamp indicating that the CE is complete only after the final IE) and it's seven related IE.

RESEARCH AIMS

The goal of this study was to develop and test a method for automatically detecting usability problems that could replace video analysis of think aloud protocols. The intended first use is with pre-defined tasks identical to those created for a TAU. Users will be brought into the Usability Laboratory and asked to perform the tasks much as they would for a TAU, but without thinking aloud. Automated data analysis will produce a report of usability problems based on these tasks without the need for video coding. To accomplish this goal we:

1. developed a simple algorithm that reproduces almost all of the video-coding criteria used in detecting usability problems with TAU, which exploits our generic method for representing interface events and client events.
2. tested the AUA algorithm by comparing the usability problems identified by traditional TAU and AUA in a single set of users.
3. calculated a correction co-efficient to account for the known slowing effect of thinking aloud and modified the algorithm appropriately.

MATERIALS AND METHODS

Subjects: The study was approved by the University of Pittsburgh Institutional Review Board (IRB Protocol #020348). Subjects were medical students with no previous experience using the SlideTutor system. Subjects were solicited by email. They were divided into two matched groups each consisting of 5 students:

Group I was trained to provide think aloud usability protocols using standard methods [3]. Briefly, subjects (1) received a standard set of instructions for thinking out loud, (2) listened as the researcher demonstrated thinking aloud on a multicolumn addition problem, (3) listened as the researcher demonstrated thinking aloud while changing margins in MS Word to 1", and (4) practiced thinking aloud as they changed the paper orientation in MS Word from Portrait to Landscape. Subjects who demonstrated difficulty in producing protocol material of sufficient quality or quantity were given feedback and asked to try the practice again. *Group II* was asked to use the system without think aloud instructions. Neither group was trained to use the interface.

Data Collection: At the start of the session, subjects were given a set of index cards, each with one printed task description. There were a total of 6 tasks designed to cover most of the functionality of the system including repetitions to examine usability problems over time. Subjects were instructed to complete each task in order. Group I was prompted to return to think aloud if the subject was silent for more than one minute. For group I, digital audio was captured concurrently with video screen capture using Camtasia Recorder v 3.0.2, and saved as .avi files for coding and review.

System: For this study, we used the case-based (node and arc) SlideTutor interface. A demonstration version is accessible at http://slidetutor.upmc.edu/.

Coding of Usability Problems: We employed a widely accepted metric for coding of usability problems in TAU[10]. Each task was subdivided into component steps, and then we applied standard criteria for establishing a usability problem. These criteria are: (1) the task could not be successfully completed (2) the amount of time to complete any individual sub-goal exceeded 2 minutes (3) the user had to try three or more things to fulfill a specific subgoal and (4) the user expressed negative affect such as frustration or anger, or the user offered a suggestion for a design change. Any subgoal that fulfills one or more criteria above for any user is considered a usability problem. During audio/video playback, we coded Group I for all actions taken according to these criteria.

RESULTS

Algorithm

We were able to account for criteria 1-3 (but obviously not #4) with our algorithm, shown below in pseudocode:

```
forall users
  forall tasks
    forall pre-defined subgoal (SG_TYPE , SG_LABEL)
      IF there is no CE matching the subgoal (CE_TYPE =
            SG_TYPE and CE_LABEL = SG_LABEL), OR
      IF timestamp $CE\_IE_{last}$ – timestamp $CE\_IE_{first}$ > 2
            minutes * $\kappa$ (where $CE\_IE_{last}$ and $CE\_IE_{first}$ are
            the last and first IE associated with a CE
            matching the SG and $\kappa$ is the correction
            coefficient for non-verbalization conditions), OR
      IF   total number of CE_IE – expected SG_IE >3
      THEN a usability problem is present involving SG
```

Comparison of data from TAU and AUA in Group I to validate the use of automated data

We identified 12 total usability problems using TAU in Group I (Table 3). Our algorithm correctly identified all 10 usability problems fulfilling criteria 1-3, and therefore missed 2 usability problems. Problems identified included use of hint buttons, use of support links and refute links when evidence or hypothesis nodes are overlapping, describing feature qualities, using the glossary, and traversing the Space-Tree based browser.

Think Aloud Usability Analysis (TAU) of Group I	
Number of tasks fulfilling each of the manual coding criteria	
Failed to complete task	2
Individual SG requiring > 2 minutes to complete	3
Attempt to complete SG required attempting >3 different things	5
Expression of negative affect or design suggestion	2
Total number of usability problems detected	12

Table 3: TAU results from Group I

Calculation of a correction coefficient (κ)

Think-aloud processes associated with recoding of sensory stimuli into verbal processes have been called Level 2 verbalization [11]. TAU is an example of Level 2 verbalization because of the abundant visual stimuli associated with using a computer interface. Level 2 verbalization has been shown to increase the time required for task completion, without significantly affecting task performance. Because we want to use AUA without requiring users to think aloud – we needed to determine the relative slowing effect of thinking aloud in order to correct for it in our algorithm.

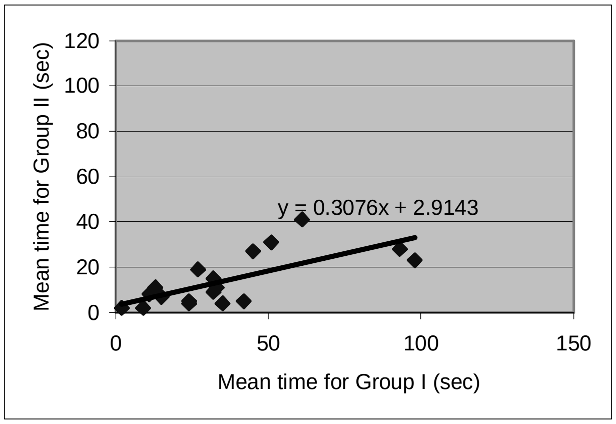

Figure 2. Effect of thinking aloud on time to subgoal completion

Figure 2 shows the mean time to completion of each subgoal for Group I versus Group 2. Each point represents one of the 19 subgoals associated with the 6 tasks. The slope of the line shows that students in the think-aloud condition (Group I) took an average of nearly three times as long to complete subgoals than those who did not think aloud. (Group II). We use the value 0.3 as a correction co-efficient to account for the faster performance times for users who do not think-aloud. When. users are asked to think-aloud, κ should be set to 1.

Use of the AUA Algorithm to identify usability problems in the non-think-aloud group

Finally, we applied the AUA algorithm (with the addition of the correction coefficient for non-verbalizing conditions) to the IE and CE data generated automatically and stored in the protocol database, and identified a total of 10 usability problems in Group II. Many of these problems overlapped with those identified in Group I. Differences between Groups I and II reflect the distinct composition of users in these two groups.

DISCUSSION

Usability evaluation techniques have proven to be invaluable tools for assessing the quality of software. However, these methods are often difficult, time consuming and expensive. Hence, in the last decade usability research has focused on identifying the best methods to apply, determining the minimal number of subjects that must be tested, and providing tools for conducting evaluation 'at a discount' [4,6]. Video-based evaluations as TAU tend to produce massive amounts of data that can be expensive to analyze. The ratio of time spent in analysis versus the duration of the sessions being analyzed has been known to reach 10:1 [12]. Moreover, video-based evaluations can make subjects self-conscious and affect their performance [3]. Usability specialists at Microsoft, Apple and SunSoft all report the use of tools to track UI events [4,13].

We have developed and tested a simple method for capturing and storing massive amounts of interaction data for our Intelligent Tutoring System. Interface events are stored in relationship to higher order subgoals. The use of the algorithm on logged data yielded the same usability results as the traditional TAU. Although our system uses the Agent Communication Language, the approach is possible with any system (including web-based systems or vendor systems) as long as they capture and provide access to low – level interface events [14]. Implementation of the tracking mechanisms and schema required additional effort and resources, but once these low-level interface events are captured, analysis can be automated. The method is flexible to changes in the UI and therefore much more efficient in the long run than extracting information from usage logs, especially when the UI is expected to change frequently. The only significant limitation that we encountered was that automated data could not capture the affect of the user or verbalized suggestions for changes. These could be manually added by slightly modifying our interface so that an experimenter could easily indicate such an event interactively to the database.

FUTURE WORK

We are encouraged by our validation results and intend to fully implement the algorithm, anticipating that it will greatly simplify collection, analysis and reporting of usability problems in our laboratory. Usability problems identified through automated methods will be automatically entered into our Bugzilla database, and triaged to a developer for review.

Complete automation of this algorithm will require us to store simple task templates that indicate the type and label of subgoals against which CE and EI must match. Such task metadata will be added to our project database to accommodate this requirement.

Eventually, we are interested in extending these methods for identifying or predicting usability problems from client user interface events that do not require accrual of subjects specifically for the purpose of usability testing. Interface events that occur during natural usage would provide an attractive alternative, because they measure problems that occur with real usage under field conditions.

ACKNOWLEDGEMENTS

This work was supported by a grant from the National Library of Medicine (R01 LM007891-01). The authors wish to thank Eugene Tseytlin for his help with the project.

REFERENCES

1. Jokela T, Iivari N, Matero J, and Karukka J . The standard of user-centered design and the standard definition of usability: analyzing ISO 13407 against ISO 9241-11: Proceedings of the Latin American conference on Human-computer interaction, August 2003.
2. Nielsen, J. Usability Engineering. Boston: Academic Press/AP Professional, Cambridge, MA. 1993a.
3. Nielsen J, Clemmensen T, Yssing C. Getting access to what goes on in people's heads? reflections on the think-aloud technique. Proceedings of the second Nordic conference on Human-computer interaction, October 2002.
4. Hilbert D M, Redmiles D F. Extracting usability information from user interface events ACM Computing Surveys (CSUR), December 2000: Volume 32 Issue 4.
5. Shneiderman, B. Designing the User Interface. Reading, MA: Addison-Wesley, 1987.
6. Kjeldskov J, Skov M. B., Stage J. Instant data analysis: conducting usability evaluations in a day Proceedings of the third Nordic conference on Human-computer interaction, October 2004.
7. Ivory, M. Y., Hearst M. A. The state of the art in automating usability evaluation of user interfaces ACM Computing Surveys (CSUR), December 2001: Volume 33 Issue 4.
8. Crowley RS and Medvedeva OP. General Architecture for Intelligent Tutoring of Diagnostic Classification Problem Solving. Proc AMIA Symp, 2003: 185-189.
9. The Foundation for Intelligent Physical Agents. At http://www.fipa.org/
10. John, B. E., & Mashyna, M. M. (1997) Evaluating a Multimedia Authoring Tool with Cognitive Walkthrough and Think-Aloud User Studies. Journal of the American Society of Information Science, 48 (9) pp. 1004-1022.
11. Ericsson KA and Simon HA. Protocol Analysis: Verbal Reports as Data. Revised Edition. MIT Press, Cambridge, MA, 1999.
12. Weiler, P. Software for the usability lab: a sampling of current tools. In Proceedings of INTERCHI'93, 1993.
13. Chang E. and Dillon, T. S. Automated usability testing. In Proceedings of INTERACT '97, 1997.
14. Medvedeva OP, Chavan G, Crowley RS. A data collection framework for capturing ITS data based on an agent communication standard. Workshop Proceedings of the American Association for Artificial Intelligence, 2005.